RIMAC: A Novel Redundancy-based Hierarchical Cache Architecture for Energy Efficient, High Performance Storage Systems

Xiaoyu Yao and Jun Wang
Department of Computer Science and Engineering
University of Nebraska-Lincoln, Lincoln, NE 68588-0115
{xyao,wang}@cse.unl.edu

ABSTRACT

Energy efficiency becomes increasingly important in today's high-performance storage systems. It can be challenging to save energy and improve performance at the same time in conventional (i.e. single-rotation-rate) disk-based storage systems. Most existing solutions compromise performance for energy conservation. In this paper, we propose a redundancy-based, two-level I/O cache architecture called RIMAC to address this problem. The idea of RIMAC is to enable data on the standby disk to be recovered by accessing data in the two-level I/O cache or on currently active/idle disks. At both cache and disk levels, RIMAC dynamically transforms accesses toward standby disks by exploiting parity redundancy in parity-based redundant disk arrays. Because I/O requests that require physical accesses on standby disks involve long waiting time and high power consumption for disk spin-up (tens of seconds for SCSI disks), transforming those requests to accesses in a two-level, collaborative I/O cache or on active disks can significantly improve both energy efficiency and performance.

In RIMAC, we developed i) two power-aware read request transformation schemes called Transformable Read in Cache (TRC) and Transformable Read on Disk (TRD), ii) a power-aware write request transformation policy for parity update and (iii) a second-chance parity cache replacement algorithm to improve request transformation rate. We evaluated RIMAC by augmenting a validated storage system simulator, disksim. For several real-life server traces including HP's cello99, TPC-D and SPC's search engine, RIMAC is shown to reduce energy consumption by up to 33% and simultaneously improve the average response time by up to 30%.

1. INTRODUCTION

Recent years have seen a steady growth in storage clusters and data centers. A wide variety of networked services such as web hosting, e-business and outsourced storage drive the need for deploying these large scale storage systems to reduce operational cost. Among the operational cost categories in today's data center, disk storage subsystem consumes the biggest portion, driving the cost of energy up as much as 25% annually and occupying as much as 75% of the floor space [5]. Putting idle disks into a standby state is desirable to reduce energy consumption, but it affects performance. As the performance bottleneck of data center applications tends to be in disk I/O rather than other components, it is imperative to develop energy-efficient and high-performance storage systems to satisfy Service Level Agreement (SLA) [7] of users, increase the revenue of data center operators and promote environmental conservation.

1.1 State-of-the-Art

A plethora of schemes have been proposed to *actively spin down* mobile disks to conserve energy for laptop computers. Only in recent years, researchers have started investigating energy efficient schemes for server-side multiple-disk storage systems. Since conventional high-performance disks take a long time to spin up (e.g. 20s for Seagate 15K RPM SCSI disk), the requests arriving during this period can be significantly delayed. For data center applications, it is important to enforce some forms of SLA, such as average/maximum response time or throughput [7]. Therefore, it is desirable that energy-efficient policies for data centers can avoid extra delay and power consumption caused by *passive spin-up* as much as possible. (A *passive spin-up* means a standby disk has to be spun up to serve incoming requests.) Experimental results from Gurumurthi *et al.* [15] and Zhu *etc al.* [28, 27] also correlate with this observation.

A categorization of the closely related investigations is summarized in Table 1, in terms of whether (i) the schemes work for conventional disks, (ii) they focus on Just a Bunch of Disks (JBOD) without regard to the storage system organization(e.g. parity disk array), (iii) they consider effects of power aware storage cache, and iv) they introduce overhead.

Some researchers proposed new disk models to deal with the expensive spin-up overhead in high performance disks. Carrera [9] and Gurumurthi [14] *etc al.* proposed a novel multi-speed disk architecture called DRPM to dynamically

Table 1: Comparison of most related studies to this work

Name	Conventional Disk?	Parity Array?	Storage Cache?	Overhead
DRPM[14]	No	Yes	No	Not Commercialized
MAID[13]	Yes	No	Yes	Additional cache disks, Movement
PDC[22]	Yes	No	No	Unscalable performance, Movement
PA[28]/PB[29]	No	No	Yes	Locality Maintenance
EERAID [19]	No	Yes	No	Computation
Hibernator [27]	No	Yes	No	Movement
FS2 [17]	Yes	No	No	Space

adjust the disk rotation speed for energy saving. This scheme shares a similar idea with dynamic frequency scaling technique in low-power CPUs. But the mechanical difference between the CPU and disk prohibits DRPM disks from being mass-produced.

Other researchers resort to load concentration techniques to avoid passive disk spin-up. Researchers from the University of Illinois developed power-aware cache allocation/partition scheme called PB [29] and replacement algorithms called PA [28] to concentrate load toward low speed disk onto storage cache, thus achieving good energy efficiency and performance. Their solutions are based on multi-speed disks in which data can still be retrieved from low speed disks. When applied to a storage system based on single-speed disk, the power-aware cache allocation/partition scheme cannot work, as data cannot be easily read from any standby disk, even if we could aggressively allocate a larger cache space for them. In addition, allocating more cache space for standby disks (usually put in standby state due to fewer accesses) may prevent data with better locality from being placed into storage cache.

The Massive Array of Idle Disks (MAID) [13] and the Popular Data Concentration scheme (PDC) [22] concentrate the load on some disks out of a whole disk array by exploiting either block-level temporal locality or file-level popularity to conserve energy. However, neither can work for a data center environment. MAID utilizes cache disks to concentrate workload, but it also introduces extra cache disks as overhead. PDC has performance scalability problems due to its inflexible load concentration design, i.e., its performance is unscalable even when all disks are running in active mode [27].

Recently, Zhu *et al.* proposed a flexible disk reshuffle scheme with performance guarantee called Hibernator [27] to address the problem of previous inflexible load concentration schemes for parity disk arrays of multi-speed disks. It is an effective, coarse-grain power and performance management solution for multiple-speed disk based storage systems. But it still cannot thoroughly solve the passive spin-up problem of conventional disk based storage systems deployed in today's data centers.

Youssef [25] investigated the design issues of low-power disk arrays for mobile computers. However, his schemes were designed and evaluated for mobile computer workload. It is not clear whether they are workable for a data center environment.

EERAID [19] exploited redundancy in a RAID [21] system to save energy by concentrating read requests on standby disks to active disks. But their schemes are based on multi-speed disks and are solely dependent on disk-level request transformation. Experimental results indicate that its performance degradation is non-negligible.

Huang *et al.* proposed FS2 [17] to utilize free space on the disk to make replicas for hot data blocks at runtime to improve performance and save energy for the storage system. Their scheme trades disk space for performance and energy consumption. Therefore, its effectiveness is circumscribed by the available free space and the access locality in the running environment.

Among the six categories shown in Table 1, only MAID, PDC and FS2 are designed and evaluated for conventional disk based storage systems. But all of them involve one or two forms of overhead as shown in Table 1. More importantly, there is no solution that can exclusively avoid the passive spin-ups for conventional disk based storage systems without a performance penalty, even though EERAID can avoid the low-speed DRPM-disk accesses in some cases.

1.2 Motivation

In conventional disk based storage systems, there are three sources of passive spin-ups: i) *Non-blocking Read* requests that cannot be deferred in a server environment; ii) *Derivative Read* due to parity updates for write requests in the parity disk array, which cannot be deferred for data reliability (*Derivative Read* is defined as those read requests incurred by parity update in parity disk array) and iii) dirty block flushing to enforce coherency between cache and disks.

Zhu *et al.* thoroughly investigated the impacts of the *third* type of passive spin-up with write-back and write-through policies [28]. Most of the aforementioned load concentration techniques [13, 22, 28, 29, 19, 27] partially solve the *first* type of passive spin-up. But they rely on the locality of access patterns, which may involve unpredictable overheads. To the best of our knowledge, no one has studied the impact of the *second* type of passive spin-up on energy consumption. Therefore, we are striving to solve the first two types of passive spin-ups, i.e. *Non-blocking Read* and *Derivative Read* for better energy saving in this paper.

A large storage cache is widely deployed in modern storage system to improve performance. Existing work on multi-level I/O cache architectures [26] [24] [12] [11] emphasize the collaboration between storage client cache and storage server cache to improve performance. Recent work on a single level power-aware storage cache called PA-LRU [28] has proved that performance oriented OPT algorithm does not necessarily lead to energy efficiency. However, the PA-LRU [28] and PB-LRU [29] algorithms cannot be directly applied to conventional disk based storage systems for the reasons discussed before. Modern hardware disk arrays are usually equipped with a Non-volatile RAM (or battery supported SDRAM) based onboard controller cache. The size of the controller cache is considerable, ranging from 64 MB to 512 MB. More importantly, it contains parity redundancy information which can be exploited to improve energy efficiency. Unfortunately, there is no effective collaboration mechanism between the storage cache and controller caches to conserve energy in current storage systems.

These observations motivate us to reorganize the multi-level caches in storage systems such that they can collaborate with each other with awareness of the redundancy and power state of underlying disks. The objective is to solve the non-blocking read and derivative read problem for conventional disk based storage system without passive spin-up overhead, thus achieving both better performance and energy savings.

1.3 Our Contribution

In this paper, we make several contributions toward energy-efficient , high-performance storage system based on conventional disks. As a small-scale, striped disk array with parity is an important building block in modern storage systems[1], we choose small-scale RAID-5 as the focus of this paper. We summarize our novel contributions, and outline the paper:

1. (Section 3) We design a redundancy-based, two-level collaborative I/O cache architecture called RIMAC. The key idea of RIMAC is to provide an effective collaboration mechanism between the storage cache and RAID controller cache, which facilitates redundancy exploitation during power-aware request transformations (see details in Section 4) *both* in two-level I/O caches and on physical disks.

2. (Section 4) Based on the RIMAC architecture, we develop two power-aware read request transformation schemes — Transformable Read in Cache (**TRC**) and Transformable Read on Disk (**TRD**), and a power-aware write request transformation policy for parity update to address the passive spin-up problem in conventional disk based storage systems. By applying these request transformation schemes, non-blocking reads and derivative reads to the standby disk can be satisfied either directly by a hit in the storage cache, or indirectly by performing an on-the-fly XOR calculation with i) data from the storage cache, ii) parity from the controller cache, and iii) data or parity from non-standby disk(s) only if needed. As a consequence, the low power disks can stay in the standby state for a longer period of time for better energy efficiency. Better performance is achieved by avoiding the long delay of passive spin-up and unnecessary disk accesses.

3. (Section 5) We develop a second-chance parity cache replacement algorithm to improve the success rate of the power-aware request transformation polices.

[1]e.g., IBM Shark and EMC Symmetrix all employ small-scale RAID-5 ($\leq$ 8 disks) as building blocks.

We develop a trace-driven, RAID based storage system simulator to evaluate the energy and performance of a RIMAC-assisted storage system. Our simulator is built on top of a fully-validated simulator, disksim [8], with an integrated 3-state single-speed disk power model used in reference [15]. Several real-life modern storage server traces, including cello99, TPC-D and the Storage Performance Council's search engine, are used for performance evaluation. Simulation results show that RIMAC is able to achieve energy savings by up to 33% and simultaneously reduce the average response time by up to 30%, in comparison to existing systems.

2. BACKGROUND

2.1 Disk Power Model

Magnetic disks are the most widely used storage media in current large-scale storage systems. The rotation speed, to a large extent, determines both performance and energy consumption for a single hard drive. High-end storage systems typically employ high spin rate SCSI disks, ranging from 10,000 to 15,000 rotation per minute (RPM), while low-end server-class disks revolve from 5,400 RPM to 7,200 RPM. The faster the hard drive spins, the higher the energy consumption and the longer the latency for a passive spin-up. Main-stream server-class SCSI disks support three operational states: Active, Idle and Standby. A set of threshold values is used to determine the disk power mode switch.

2.2 RAID based Storage Systems

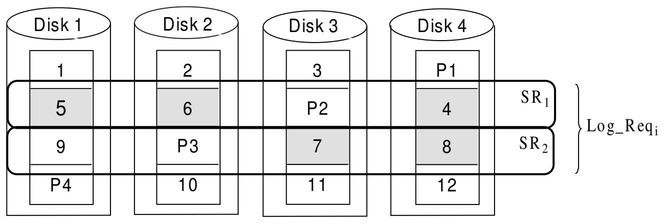

Figure 1: Illustration of Stripe Request Mapping. In a 4-disk RAID-5, each number represents one data stripe unit. And P1, P2, P3, P4 represent parity stripe units. Log_Req_i is mapped to two stripe requests: $SR_1 = \{4, 5, 6\}$ and $SR_2 = \{7, 8\}$). Unit 7, 8 are called Requested Units of SR_2 and unit 9 is called Complementary Unit of SR_2.

RAID-5 and RAID-1 are two major building blocks in large-scale storage systems such as data centers and storage clusters. For example, the EMC Symmetrix 8000 [16] holds up to 384 disks, which are protected either via mirroring (RAID-1) or a variation of RAID-5 that EMC calls RAID-S. The RAID-S group size is four or eight with 73 GB per drive.

RAID-5 is a block-interleaved disk array with distributed parity. Because the parity is evenly distributed, no disk bears excessive I/O load as the parity disk in RAID-4. A set of logically contiguous blocks that exist on each disk before allocations are made from the next disk in the array is called *Stripe Unit, or Unit.* There are two types of stripe unit in RAID-5: data stripe unit which saves user data, and parity stripe unit which stores parity data. A group of data stripe units over which a parity stripe unit is computed is called a *Stripe, or Parity Group* [10]. A logical disk request is mapped to several *Stripe Requests (SR).* In this paper, SR is defined as a subset of *requested* physical stripe units that falls into the same stripe. Given a stripe, we call the stripe unit requested by users the *Requested Units*, while the stripe unit not requested by users is the *Complementary Unit.* Figure 1 illustrates how a logical request is mapped in a 4-disk RAID-5.

3. ARCHITECTURE OVERVIEW

We develop a redundancy-based, two-level I/O cache architecture for RAID-5 based storage systems called **RIMAC**, with its architecture illustrated in Figure 2. In RIMAC, we manage the large storage cache in the unit of low-level data stripe unit and save *only* parity stripe units in a small RAID controller cache for efficiency. We interchangeably use parity cache and controller cache in the RIMAC design for the rest of the paper. Based on the protection schemes of today's real storage products and systems [2, 3], we assume that the memory used for storage cache is UPS supported and the memory used for parity cache is Non-volatile RAM (or battery supported SDRAM) to protect data reliability and recoverability.

A new *Cache Service Provider (CSP)* is introduced to handle incoming requests in RIMAC-assisted storage systems. The specific functions of three components in CSP are summarized as follows.

- The *Front-end* of CSP accepts the user requests arriving at the storage system and maintains state for outstanding requests.

- The *Bottom-Half* of CSP resides in each RAID controller acting as an agent for RAID controller cache. It provides a stripe-based request mapping interface for storage cache. It also monitors, reports and controls the power modes of underlying disks. Our power-aware write request transformation procedure (detailed in Section 4.2) gets executed here only if any write leads to standby disk access.

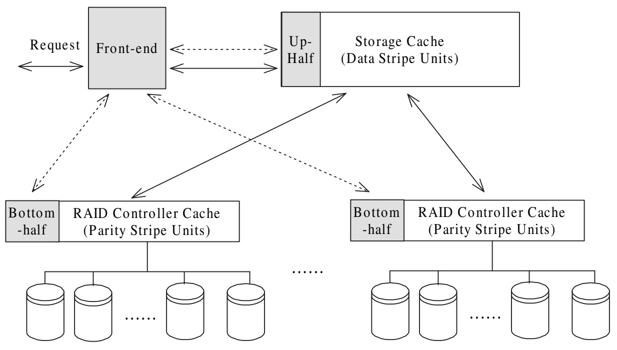
Request
Front-end
Up-Half
Storage Cache (Data Stripe Units)
Bottom -half
RAID Controller Cache (Parity Stripe Units)
Bottom -half
RAID Controller Cache (Parity Stripe Units)
......
......
......

Figure 2: RIMAC Architecture Overview. The shaded rectangles represent CSP Front-end, Bottom-Half and Up-Half of RIMAC.

- The *Up-Half* part of CSP resides in the storage cache acting as an agent for storage cache management. Data stripe units that pass through the storage system are only cached in storage cache. Our read request transformation procedure (detailed in Section 4.1) gets executed here only if any read leads to standby disk access.

The Front-end and Up-Half are implemented as software modules in a storage server. The Bottom-Half can be implemented by modifying the firmware of a modularized storage enclosure that usually incorporates a microprocessor, XOR engine, NVRAM cache and bus controllers. In our simulator, the Bottom-Half module implementation is about 2000 lines of C code. In this paper, we assume the back-end storage enclosures are connected to the storage server via SCSI buses. Many other alternatives exist for packaging RIMAC, such as high speed switches (e.g. FC switches), but are beyond the scope of this paper.

The dotted lines in Figure 2 denote the control messages among three components. Control messages are mainly used to transfer the cache lookup result and selected power-aware policy to serve the request. The solid lines denote the data transferred from and to the storage system. For read requests, the stripe units that hit in storage cache are returned directly. When our request transformation procedures get executed, the parity stripe unit is moved to Up-Half and XORed with necessary data stripe units to compute the missed stripe unit. The computational overhead of XOR calculations is minimal as most of current storage systems employ multi-way powerful CPUs as the processing unit. The time for a simple XOR calculation is far less than that of reading data blocks from disks. Compared with the power consumption of several hundreds of disks(10.2 W * 100 disks), the power consumption of an XOR-engine is negligible. For example, the power consumption of an I/O processor (IBM Power PC 750) used in EMC Symmetrix 8000 [16] is only 50 mW [6].

Figure 3 describes a sample scenario where the data stripe unit 6 on standby disk 2 is requested by a read request. As illustrated in Figure 3, RIMAC can serve this request without any passive disk spin-up.

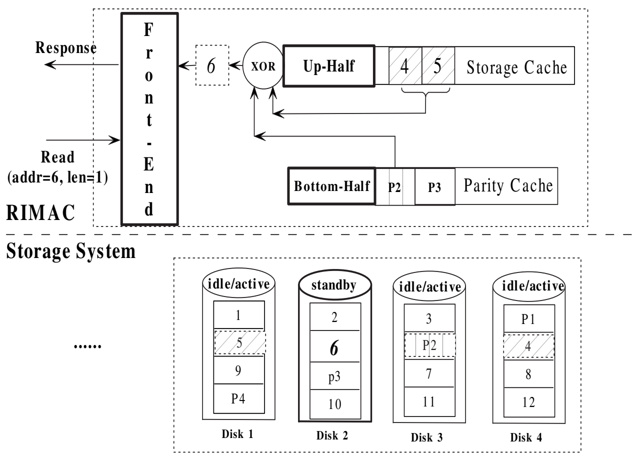

Figure 3: Data stripe unit 6 in a 4-disk parity disk array is requested by a read request. Because it is missed in storage cache, traditional schemes need a passive spin-up for Disk 2 to satisfy the request. RIMAC can serve the request by XORing stripe units 4, 5 (hit in storage cache) and p2 (hit in parity cache) without any passive spin-up at all.

There are several advantages behind the RIMAC design. First, for the sake of energy-efficiency, RIMAC provides a better opportunity to perform power-aware request transformation in a two-level collaborative I/O cache environment. Second, for the sake of performance optimization, an exclusive two-level cache design that stores data stripes in storage cache and parity stripes in controller cache can save space because the current RAID controller cache size (usually larger than 128 MB [3]) is non-negligible compared with storage cache size (usually several Gigabytes [16]). Third, a parity exclusive controller cache design allows RIMAC to store user data closer to the user and parity closer to RAID, which avoids the data coherency and movement between storage cache and RAID controller cache. Last, non-blocking read and derivative read requests toward standby disk can be satisfied by XORing data stripe units from the storage cache and parity stripe unit from controller cache, which improve both energy efficiency and performance.

4. POWER AWARE REQUEST TRANSFORMATION

The design goal of RIMAC is to save energy by avoiding any uncontrollable disk spin-up and spin-down derived from non-blocking reads or derivative reads, and to improve performance by minimizing the total number of disk accesses and queueing delay. Because of the limited redundancy ratio maintained in RAID-5, we can only have at most one disk in standby state in order to perform request transformation successfully. The theoretical upper bound of energy savings in a RIMAC-assisted N-disk RAID-5 is $\frac{1}{N}$. As large-scale storage systems adopt 4 to 8 disk RAID-5 as the building block, spinning down one disk per small-scale RAID-5 is still able to achieve good overall energy savings.

The main idea of RIMAC is to exploit power-aware two-level I/O cache architecture and its redundancy-awareness to save energy and simultaneously improve performance by applying **Request Transformation** both in two-level I/O cache and on disks for read and write requests. We summarize our request transformation policies in Table 2.

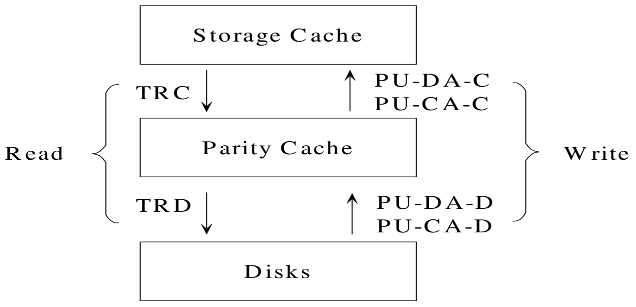

Figure 4: Power Aware Request Transformation Policies in RIMAC.

As indicated in Figure 4, for read requests, our request transformation follows the left arrow in a top-down manner triggered by the Up-Half of CSP; namely, the storage cache uses the redundancy information in parity cache to serve read requests for the standby disk. For write requests, our request transformation follows the right arrow in a bottom-up manner triggered by the Bottom-Half of CSP; namely, the parity cache utilizes higher level storage cache information to fulfill parity update with power awareness. Given such a two-way collaborative cache design for both reads and writes, RIMAC is able to exploit the inherent redundancy at maximum to both save energy and improve performance.

4.1 Read Request Transformation

For read, we design two power-aware schemes, *Transformable Read in Cache (TRC)* and *Transformable Read on Disk (TRD)*, by taking storage cache, controller cache and the power state of underlying disks into consideration. By applying TRC and TRD, we optimize the disk I/O access pattern within the array such that the idle period of the standby disk can be elongated for significant energy savings while the overall storage system performance can be improved due to additional hits resulting from TRC.

4.1.1 Transformable Read in Cache

The idea behind TRC is straightforward. Assuming we have an N-disk RAID-5, given a stripe request, if there are N-1 stripe units hit in storage cache and controller cache in total, the missed stripe unit can be generated on-the-fly by XORing all the hit stripe units without any disk access. This is a *TRC hit*. The condition for TRC is defined by Formula 1.

$$\begin{aligned} N-1 &= Num(Hit\_DataStripeUnits) \\ &\quad + Num(Hit\_ParityStripeUnit) \end{aligned} \tag{1}$$

Type	Level	Policy	Notes	Section
Read	Cache	TRC	Transformable Read in Cache	4.1.1
	Disk	TRD	Transformable Read on Disk	4.1.2
Write	Cache	PU-DA-C	Parity Update with Direct Access in Cache	4.2
	Cache	PU-CA-C	Parity Update with Complementary Access in Cache	4.2
	Disk	PU-DA-D	Parity Update with Direct Access on Disk	4.2
	Disk	PU-CA-D	Parity Update with Complementary Access on Disk	4.2

Table 2: Summary of Request Transformation Policies

The advantage of TRC is that even though one disk is temporarily in a standby state, the requests can still be served without spinning up the disk, reducing the spin-up and spin-down frequency. Even if the missed stripe unit is on an active or idle disk, we can still save one physical disk access.

4.1.2 Transformable Read on Disk

TRC is kills two birds with one stone, saving energy and improving performance. But if there is more than one data stripe units missing for a stripe request, we need to use a Transformable Read on Disk (TRD) policy. In essence, TRD shares a similar idea with TRC in exploiting redundancy to redirect standby disk accesses. We formulate the condition of TRD in Formula 2.

$$\begin{aligned} N-1 = {} & Num(Hit\_DataStripeUnits) \\ & +Num(Hit\_ParityStripeUnit) \qquad (2) \\ & +Num(Missed\_DataStripeUnits\ on\ NonStandbyDisk) \\ & +Num(Missed\_ParityStripeUnit\ on\ NonStandbyDisk) \end{aligned}$$

TRD is applied when the condition of TRC cannot be satisfied. Due to the non-blocking nature of a read request, a TPM (Traditional Power Management) based system [14] has to read the data from the requested disks to get the missing stripe units. In this paper, we call this policy *Direct Access (DA)*. If the stripe unit is on a standby disk, TPM has to spin up the standby disk before serving the request, which usually takes several seconds and causes non-negligible energy consumption. However, the TRD policy can bypass accesses to the standby disk by generating the requested data through XORing both requested and complementary (non-requested) data stripe units from the same stripe, which are available either in storage cache or on non-standby disks. Given the assumption that one disk per RAID-5 can be spun down to standby state, TRD always acts as a reliable backup for TRC for energy-efficiency. When both TRD and DA policies do not access the standby disk, the one that results in a smaller number of disk accesses is chosen to run.

4.2 Write Request Transformation

For writes, a write-allocate, write-back policy is adopted to reduce energy consumption without significant performance degradation. The energy impact of write back and write through policies was fully studied in previous work [28] with emphasis on parallel disk systems. However, the energy impact of parity updates caused by writes is not well addressed. This has to be thoroughly researched because distinct parity update polices may result in different numbers of disk I/Os, and thus different amounts of energy consumption and performance variation. Youssef [25] proposed a dynamic remapping scheme using a log-based allocation strategy for write and deferred parity updates to save energy for disk array in laptops. However, his solution works well for RAID-4 rather than RAID-5.

To solve the problem, we develop a novel write request transformation policy for parity update named *Parity Update with Power Awareness(PUPA)*. Since the PUPA can be accomplished either in caches or on disks, we classify the parity update scheme of PUPA into four sub-classes:

1. *PU-DA-C*: update parity by XORing old requested data stripe units, old parity stripe unit and the newly written data stripe units. Here all old data stripe units and parity stripe unit are solely from storage cache and parity cache;

2. *PU-CA-C*: update parity by XORing complementary data stripe units with newly written data stripe units. Here all the complementary stripe units are solely from storage cache;

3. *PU-DA-D*: similar to PU-DA-C policy, update parity with some of the requested old stripe units from non-standby disks;

4. *PU-CA-D*: similar to PU-CA-C policy, update parity with some of the complementary stripe units from non-standby disks.

PU-MIN (parity update with minimum disk I/Os) is a procedure used to update parity by choosing either PU-DA-D or PU-CA-D. The one with the smaller number of physical disk accesses involved is chosen to run for better performance.

The PUPA algorithm with four parity update schemes is detailed in Algorithm 1 and Algorithm 2. Both algorithms work for every stripe request SR_i mapped from a logical request *Log_Req*. The output is the sub-policy chosen by our power-aware parity update algorithm based on the cache lookup result and the power state of underlying disks. According to the chosen policy, CSP reorganizes the derivative read requests and dispatches them to the corresponding disks. Once finished, the Bottom-Half XORs all the necessary stripe units and updates the parity immediately. Algorithm 1 shows how the PUPA algorithm makes decisions when all the requested data stripe units hit in the storage cache.

Algorithm 1: Full Requested Stripe Units Hit Case

```
Data: $SR_i$ (Stripe Request i).
Result: Power-aware parity update policy.
begin
    if Old Parity HIT then
        return PU-DA-C
    else if Old Parity MISS then
        if all Complementary Stripe Units HIT then
            return PU-CA-C
        if Old Parity not on standby disk then
            return PU-DA-D
        if Old Parity on standby disk then
            return PU-CA-D
end
```

A more complex case about partial hits of stripe request(i.e., some of the requested stripe units hit in storage cache) is detailed in Algorithm 2. During the warmup phase, Algorithm 2 can also deal with the full stripe request miss case (i.e., none of the requested stripe units hit in storage cache).

Algorithm 2: Partial Requested Stripe Units Hit Case

```
Data: $SR_i$ (Stripe Request i).
Result: Power-aware parity update policy.
begin
    if Old Parity HIT then
        if Missed Requested Stripe Units on standby disk then
            return PU-CA-D
        else
            if Missed Complementary Stripe Units on standby disk then
                return PU-DA-D
            else
                return PU-MIN
    else if Old Parity MISS then
        if Missed Old Parity on standby disk then
            return PU-CA-D
        else
            if Missed Requested Stripe Units on standby disk then
                return PU-CA-D
            else
                return PU-MIN
end
```

5. DESIGN ISSUES

5.1 Cache Placement and Replacement

The cache hit rate and power-aware request transformation rate of RIMAC are directly related to its two-level cache placement and replacement policies.

Storage Cache: Our storage cache placement policy is request based; namely, only the user-requested data stripe units are placed into the storage cache. To avoid cache pollution, we do not place those stripe units that are fetched from disks for request transformation into storage cache. A small buffer is used to temporarily accommodate data for XOR computation. Given a request transformation policy, RIMAC sets the upper limit of the total number of stripe units allocated per stripe in both parity cache and storage cache to be $N-1$ in an N-disk RAID-5. When making the placement decision, both storage cache and parity cache refer to the *stripe allocation counter* in Up-Half to enforce this constraint. The rationale behind this is to indirectly enlarge the effective storage cache space and fulfill more requests without increasing the physical cache size. RIMAC is orthogonal to the state-of-the-art cache replacement policies, such as LRU, LFU, ARC [20] and MQ [26]. In this paper, we choose LRU as an example for storage cache replacement.

Parity Cache: The placement operation in parity cache may be triggered by two types of events: i) TRD when the missing parity stripe unit is fetched from disk, and ii) write allocation when the parity is missing in parity cache. In both cases, the parity stripe unit is placed into parity cache. Since the request transformation and parity update optimization rely on the parity supplied by the parity cache, there is a need for developing a replacement algorithm to obtain high TRC and TRD success ratios while still maintaining good access locality. An optimal solution in theory is to replace a parity whose corresponding stripe contains the smallest number of data stripe units in storage cache. Unfortunately the runtime overhead to implement such an algorithm is prohibitively high. The MIN-Heap algorithm needs $O(log_2 n)$ time to maintain the min-heap property upon each storage cache placement and replacement, where n is the number of stripes that have data cached in storage cache.

We develop a variance of the LRU algorithm called the *second-chance parity cache replacement algorithm* to strike a tradeoff between the request transformation ratio and cache maintenance overhead. Given a parity replacement event, the algorithm checks the victim chosen by LRU and grants it a second chance, only if its stripe allocation counter is larger than or equal to $\lceil \frac{N-1}{2} \rceil$. Otherwise, the victim parity stripe unit is directly replaced. The cache maintenance overhead for the stripe allocation counter in Up-Half is $O(1)$. We only need one bit to indicate whether a second chance has been granted or not. By this way, RIMAC can potentially increase the ratio of TRC and reduce the I/O overhead when applying TRD.

5.2 Write Policy

RIMAC uses Non-volatile RAM (or battery supported SDRAM) for parity cache and UPS supported RAM for storage cache to recover from system crashes or disk failures. We adopt a write-back policy same as the one studied in reference [28]. The threshold to flush dirty blocks was set to 40% as recommended by EMC [1]. However, the destage procedure to flush dirty blocks takes the standby disk into consideration. It only flushes dirty blocks to non-standby disks and defers writing dirty blocks to the standby disk. When the dirty blocks of the standby disk exceed a certain threshold ratio of the maximum cache space allocated for dirty blocks in storage cache, the standby disk spins up to enforce reliability. Under normal workload, we set the maximum number of standby disks per small RAID-5 to be one. Therefore, holding dirty blocks for one disk per small RAID-5 will not lead to severe resource contentions for RIMAC. Our experimental work also proves this conjecture.

5.3 Hybrid Disk Power Management

To effectively adjust disk power states for large storage systems, previous research adopted an epoch based approach, choosing either a fixed number of requests called *request window* or a fixed period of time called *time window* as the epoch for disk dynamic power management [14, 28]. In RIMAC, we use a hybrid approach: current epoch duration is the shorter of 900 seconds (time window) or the duration in which 15,000 requests have been processed since the last epoch. When the offered load of an N-disk RAID-5 falls below 80% of the maximum bandwidth of a RAID-5 with (N-1) disks (assuming spinning down one disk in N-disk RAID-5), we selectively spin down the disk that has the maximum average inter-arrival time during the last epoch. Here 80% represents the saturation knee of system service capacity [9]. Although there are no physical accesses toward the standby disk in RIMAC because they are transformed into accesses in cache or on other non-standby disks, the inter-arrival time of the standby disk is estimated based on the accesses before applying request transformation policies.

RIMAC has no passive spin-up for non-blocking read and derivative read requests. Only under two extreme cases, RIMAC actively spins up a standby disk. The first is when the offered load exceeds 80% of the maximum bandwidth of a RAID-5 with (N-1) disks, which indicates that a (N-1) disk RAID-5 does not have enough capacity to service current workload. The second is when the dirty block ratio of the standby disk in storage cache exceeds certain dirty flush watermark specifically set for standby disk, which indicates that the standby disk needs to be spun up to enforce reliability.

6. EVALUATION METHODOLOGY

We developed a trace-driven, RAID based storage system simulator to evaluate the performance of RIMAC-assisted storage system, with about 5000 lines of C code to implement CSP Front-end, Up-Half and Bottom-Half. A validated simulator, disksim [8], is extended with 1000 lines of C code for a three-state single speed disk power model ported from a reference work [15] and interfaces between RIMAC and disksim. We simulated a 40-MHz 16-bit Ultra160 SCSI bus in disksim to connect the Up-half and the Bottom-half. Theoretically, the maximum bandwidth (160 MB/s) can transfer an 8 KB parity stripe from Bottom-Half to the XOR engine in 0.049 ms. In our simulation, the memory chips we used for both parity cache and storage cache are DDR SDRAM PC2100, which provides a peak transfer rate of 2128 MB/s. We used an IBM Ultrastar 36Z15 high performance SCSI disk as the disk model during the experiment. The key parameters are shown in Table 3.

Parameter	Value
Standard Interface	SCSI
Capacity	18.4 GB
Rotation Speed	15000 RPM
Disk Controller Cache	4 MB
Average Seek Time	3.4 ms
Average Rotation Time	2ms
Transfer Rate	55 MB/s
Power (Active)	13.5 W
Power (Idle)	10.2 W
Power (Standby)	2.5 W
Spin Down Energy (Idle→Standby)	13 J
Spine Down Time (Idle→Standby)	1.5 s
Spin Up Energy (Standby→Active)	135 J
Spin Up Time (Standby→Active)	10.9 s

Table 3: IBM 36Z15 Disk Parameters

The RIMAC CSP components (Front-end, Bottom-Half and Up-Half) are implemented in our RIMAC simulator. All I/O requests are handled by CSP in the order of front-end parsing, cache lookup, request transformation and parity update optimizations. Once a request has been serviced, the corresponding energy consumption and response time are recorded. The response time per request is defined as the time between a logical I/O request arrives at the CSP front-end and the response of the logical request departs the CSP front-end. At the end of each epoch, the average response time and workload characteristics are reported to CSP. CSP makes disk power state adjustment decision based on the offered load and chooses a candidate disk that has a maximum average inter-arrival time.

In our experiments, we chose three real-life storage server traces: cello99 and TPC-D collected by the HP Storage Research Lab, and the Storage Performance Council's search engine trace [4] (SPC-SE in brief) to evaluate RIMAC. Cello99, TPC-D and search engine represent a modern file server, a decision-making database server and modern Web search engine applications respectively. The statistics of the traces are summarized in Table 4. Cello99 and TPC-D traces contain mixed read and write requests with different read-to-

	Cello99	TPC-D	SPC-SE
Read Ratio (MB)	57.3%	82.5%	99.9%
Avg Req_Size (KB)	12	53	15
Dataset (MB)	1023	105,075	67,395

Table 4: A summary of Trace Statistics

write ratio from multiple users, which well represent modern storage system workloads. For cello99, we extracted a one-hour trace segment with a 1,203 MB data set in total and 57.3% reads. The TPC-D trace segment was collected when running a TPC-D q10 benchmark. The SPC's search engine trace (SPC-SE) was a read-exclusive trace collected by running a popular search engine application.

We chose a five-disk RAID-5 for performance evaluation without explicit explanation in this paper as small-scale RAID is the focus. We conducted a scalability study on RIMAC by changing the RAID size from four to eight disks in Section 7.6. The stripe unit size was set to 16 KB for TPC-D, 8 KB for SPC-SE and 4 KB for cello99 because the average request size in cello99 is smaller. The baseline RAID system was configured with a storage cache and a RAID controller cache, both of which adopted an LRU cache replacement scheme. Its controller cache contained both data and parity. The cache line size of the baseline system and RIMAC for both storage cache and parity cache was set to be the same as the stripe unit size to simplify alignment handling. During the experiments, both storage cache and controller cache in the LRU baseline system and RIMAC were always configured with the same size for fair comparison. The baseline system employs a threshold-based dynamic disk power management scheme, with a threshold value set to a break-even time (two seconds) based on a competitive analysis in reference [18].

7. EXPERIMENTAL RESULTS

We replayed three traces and collected results by running simulations for two systems: the LRU baseline system (BASE for short) and RIMAC. BASE and RIMAC were configured with total cache size varying from 32 MB to 256 MB. INF is a baseline system with an infinite cache configuration. Both energy and average response time of all algorithms are normalized to that of INF.

As the working set of cello99 is smaller than that of TPC-D, we configured the total cache size to 32 MB and 64 MB for cello99, 64 MB and 128 MB for TPC-D and 128 MB and 256 MB for SPC-SE. During the experiments, the RAID controller cache size was fixed to 8 MB while the storage cache size was equal to the total cache size minus 8 MB for both systems. We also conducted a sensitivity study on the impact of parity cache size in Section 7.4.1.

7.1 System Performance

Figure 5 compares the overall average response time of BASE and RIMAC with three workloads respectively. The X axis in these figures denotes the total cache size (in MB) used in experiments. INF means an infinite cache size.

As shown in Figure 5, RIMAC consistently beats the BASE for all different cache configurations. For example, with cello99, RIMAC with a 32 MB total cache size (RIMAC-32 for short) achieves 29% improvement over BASE with the same cache size 32 MB (BASE-32 for short) in terms of average response time. Given an increasing total cache size to 64 MB, RIMAC speeds up the average response time by 30% compared with BASE-64. The reason is that the collaborative work between storage cache and controller cache in RIMAC indirectly increases the effective cache space, thus obtaining better performance than the baseline system. Even with an extreme setting, RIMAC-INF still outperforms BASE-INF by 20% in average response time. The reason is that although RIMAC-INF spins down the disk to standby to save energy, it can service the requests without bothering the standby disk and thereby reduces the number of expensive passive disk spin-ups. As a consequence, even a RIMAC-32 can achieve 6% better performance than BASE-INF.

Figure 5 (b) and Figure 5 (c) show similar trends of performance improvement by replaying TPC-D and SPC-SE traces. In terms of average response time, RIMAC improves performance from 2% to 6% in TPC-D and from 5% to 14% in SPC-SE over the baseline. As the TPC-D workload has random access patterns, the performance benefit derived from cache is lower than that of workloads with better localities, such as cello99 and SPC-SE. However, RIMAC-128 still achieves 3% better performance than LRU-INF for TPC-D. The reason is that all request transformation policies in RIMAC avoid some of the passive spin-ups. In addition, TRC, PU-CA-C, and PU-DA-C can indirectly reduce the number of cold misses in storage cache and thus further boost the performance.

7.2 Energy Consumption

Figure 6 shows the energy consumption of the three traces for RIMAC and BASE with different configurations. In addition to performance improvement, RIMAC also saves energy in serving I/O requests. As shown in Figure 6 (a), with the cello99 trace, RIMAC can consistently save 14-15% energy compared with LRU baseline systems. Figure 6 (b) and (c) show similar trends for TPC-D (33-34%) and SPC-SE (13-16%). The main reason RIMAC can save energy while simultaneously improving the performance is that RIMAC can selectively spin down a disk to standby, service the requests without bothering the standby disk and thereby reduce the number of expensive disk passive spin-ups. As a consequence, the standby disk can stay at the low power state much longer to obtain better energy efficiency without blocking any request.

In RIMAC, the key to saving energy is redundancy exploitation, which sets a theoretical upper bound for energy conservation in an N-disk RAID-5 as $\frac{1}{N}$. Here the upper bound is 20% (1/5), which matches the energy consumption results for cello99 and SPC-SE. However, the energy savings of TPC-D significantly breaks the 20% upper bound. After analysis of the traced outputs, we find that the threshold-based power management scheme used in BASE works pathologically under a random workload like TPC-D. More passive spin-ups were observed for the TPC-D work-

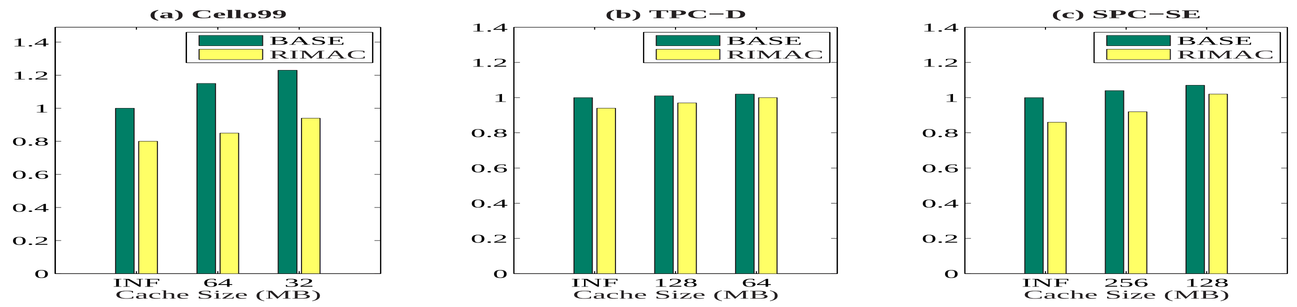

Figure 5: Comparison of normalized average response time for different workloads.

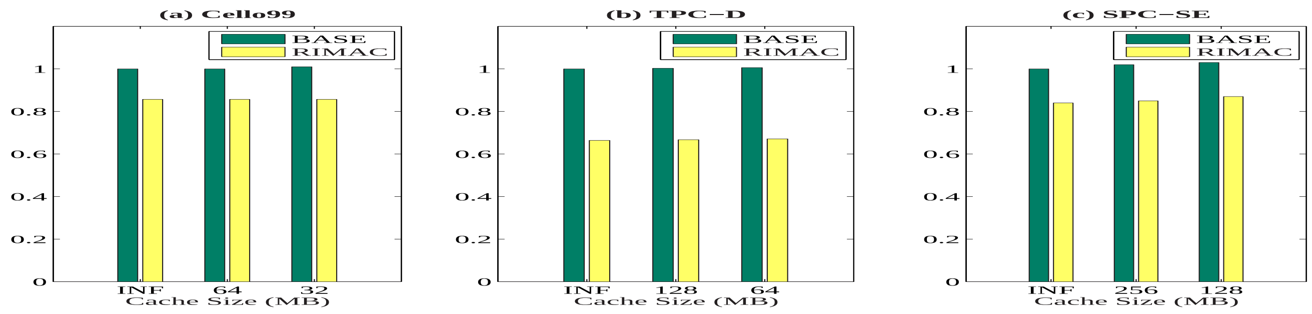

Figure 6: Comparison of normalized energy consumption for different workloads.

load, which negatively contributes to the energy efficiency of BASE. RIMAC, on the other hand, can throttle the number of passive spin-ups for better energy efficiency. But more disk access overhead due to disk level request transformations among random accesses of TPC-D prevents RIMAC from offering significant performance improvement comparable to that of cello99 and SPC-SE.

Combining the performance results, we found that increasing the cache size does improve the storage system performance but does not necessarily improve the energy efficiency in both systems. Given a proper cache size configuration, RIMAC can achieve near-optimal performance and energy efficiency comparable to that of RIMAC with an infinite cache size. This implies our algorithms work well enough.

7.3 Performance Analysis of Request Transformation

To find out why RIMAC can save energy and improve performance at the same time, we investigate the effects of the power-aware request transformation on performance improvement and energy saving in this section. We first study the effects of the read request transformation and its overhead compared with an LRU baseline system of commensurate size. Then we look into the effects of power-aware parity update policies and their overhead. Due to space limitations, we only present the results of RIMAC with a 64 MB total cache size for cello99, 128 MB for TPC-D and 256 MB for SPC-SE and the LRU baseline system of commensurate cache size in this section.

We break down the cache performance results into read-hit ratio, transformable read in cache (TRC) ratio and transformable read on disk (TRD) ratio. Since TRD introduces extra disk accesses toward non-standby disks, an *extra access (EA)* ratio is used to justify such overhead during the request transformation. Here the ratio is the fraction of all requested blocks that fall into a specific category.

7.3.1 *Effects of Read Policies*

Figure 7 (a) shows the results of a 64 MB BASE and RIMAC under the cello99 workload. For performance, RIMAC achieves a 2.1% higher read hit ratio than BASE. This increase derives from the ”stripe size minus one” constraint that indirectly increase the overall cache space. An additional 3% TRC ratio (out of the overall accesses) is observed in RIMAC. TRC can absorb requests on standby disks by two-level cache collaboration, compared with expensive passive disk spin-ups in baseline systems. In all, RIMAC achieves a 33.8% ($\frac{17.2+3-15.1}{15.1}$) higher cache hit rate than BASE. For energy conservation, RIMAC redirects 10.1% (3% + 7.1%) of all block accesses to bypass the standby disk and save energy, which is impossible to realize by existing solutions.

Figure 7 (b) shows the results of a 128 MB BASE and RIMAC under the TPC-D workload. The relatively lower cache hit ratio for both RIMAC (5%) and BASE (6.3%) is mainly caused by the random access pattern and larger request size of the TPC-D workload. Although the read hit ratio of RIMAC is 1.3% lower than that of BASE, the aggregate RIMAC cache hit rate is 0.8% higher than that of

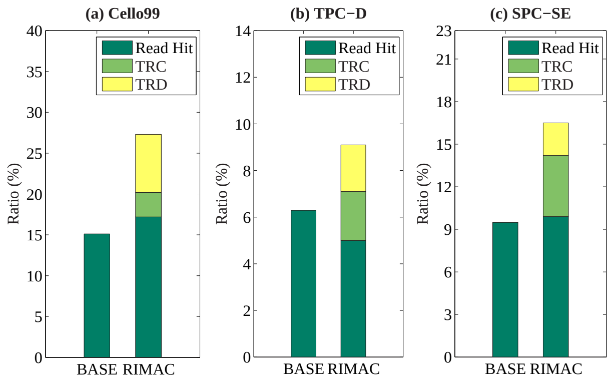

Figure 7: Read hit ratio of RIMAC vs. BASE for different workloads.

BASE, which represents 12.8% ($\frac{5+2.1-6.3}{6.3}$) improvement on the cache hit rate. In addition, RMIAC redirects 4.1% (2.1% + 2%) of all block accesses to bypass the standby disk for energy conservation.

For the read-exclusive Web search engine application, Figure 7 (c) shows that RIMAC improves the read hit ratio by 49.5% ($\frac{9.9+4.3-9.5}{9.5}$) and simultaneously redirects 6.9% of all block accesses to bypass the standby disk for energy saving.

7.3.2 Overhead of Read Policies

For those blocks that are missing from the caches, the baseline system always resorts to DA (direct access policy) which may involve expensive passive spin-ups. Both TRC and TRD in RIMAC eliminate passive spin-up overhead. But TRD introduces additional I/O requests to some of the non-standby disks. Figure 8 quantifies the overhead of read request transformation policies with the same cache settings as section 7.3.1.

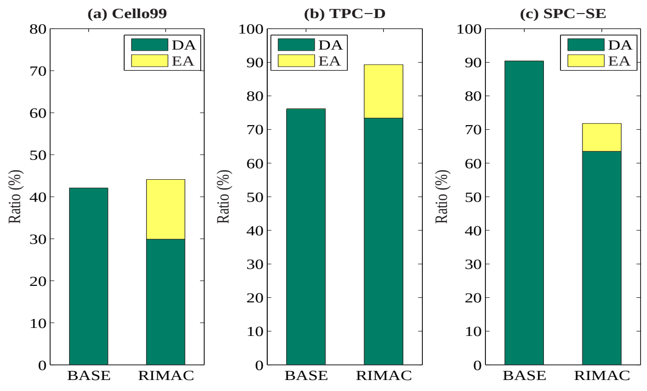

Figure 8: Read overhead of RIMAC vs. BASE for different workloads.

As shown in Figure 8 (a), for disk access, the sum of the ratio for direct access (explained before, disk access derived from cache miss without transformation) and extra access in RIMAC is only 2% (29.9%+14.2%−42.1%) higher than that of direct access in BASE, which implies our transformation schemes introduce minor disk I/O overheads. For TPC-D workload, as shown in Figure 8 (b), we observe 13.1% (73.4%+15.9%-76.2%) additional disk accesses sacrificed to bypass the standby disk. This overhead, to some extent, explains why TPC-D only achieves a small performance improvement (3%) over BASE. Figure 8 (c) shows that SPC-SE even has (18.6%) less disk accesses than BASE, which produces 8% performance improvement and 15% energy saving as discussed in Section 7.1 and 7.2.

7.3.3 Effects of Power Aware Parity Update Policy

Figure 9 compares the performance of parity update policy on RIMAC and the baseline system. Because SPC-SE is a read-exclusive workload, we only present the results of a 64 MB total cache size for cello99 and 128 MB for TPC-D. In both cases, RIMAC achieves a comparable hit ratio for data writes to that of the LRU baseline system.

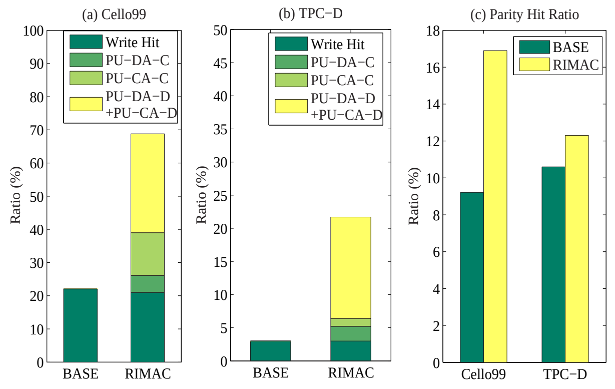

Figure 9: Write update performance of RIMAC vs. BASE for Cello99 and TPC-D.

As shown in Figure 9 (a), for cello99, the fraction of all requested blocks that are cache write hits in RIMAC is 21%. For parity update purpose, not all write hits directly lead to performance improvement and energy savings. To avoid standby disk access, we observed only 5.1% of old data that hit in storage cache can be directly used to update parity using the PU-DA-C policy. And 12.9% of writes resort to the complementary update method by reconstructing the parity using complementary stripe units from the same stripe that are in storage cache. The reason is that more parity updates in cello99 may access missed parity or old data from standby disks. Under this situation, complementary access in cache can work effectively as a backup to improve both performance and energy efficiency during parity update.

As shown in Figure 9 (b), for TPC-D, the fraction of all requested blocks that are cache write hits is only 3%. One reason may be that there are only 17.5% writes in total in the TPC-D workload while cello99 has 43% writes. In addition, most of the request size is relatively large. Therefore, the chance to apply PU-DA-C (2.2%) and PU-CA-C (1.2%) is smaller than that of cello99.

As shown in Figure 9 (c), the parity hit ratio has been improved in RIMAC by 83.7% ($\frac{16.9-9.2}{9.2}$ for cello99 and 13.8% for TPC-D, because RIMAC employs a parity-exclusive RAID controller cache design along with a second chance replacement algorithm.

7.3.4 Overhead of Power Aware Parity Update Policy

For those missing blocks necessary for a parity update, baseline system always resorts to parity update with minimum number of disk accesses (PU-MIN), assuming best performance is solely dependent on the number of disk accesses. However, this policy may not be optimal in energy-saving contexts because it may involve expensive passive spin-ups. The power-aware parity update policy in RIMAC can avoid passive spin-up overhead. But it also adds I/O requests to some of the non-standby disks. Figure 10 quantifies the overhead of the power-aware parity update policies under cello99 and TPC-D with the same settings as section 7.3.3.

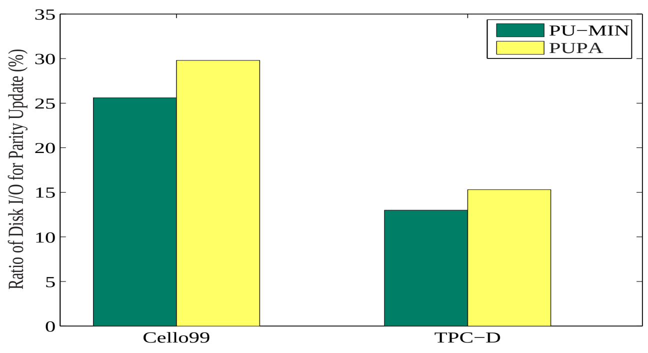

Figure 10: Parity update overhead of RIMAC vs. BASE for Cello99 and TPC-D.

As shown in Figure 10, RIMAC only increases 4.2% (cello99) and 2.3% (TPC-D) disk accesses due to the power-aware parity update, compared to a PU-MIN exclusive policy in baseline system. This cost is still far less than that of accessing a standby disk directly, which implies both the energy and performance benefits outweigh the cost of RIMAC.

7.4 Impact of Parity Cache Design in RIMAC

As Non-volatile RAM for parity cache (32 MB to 256 MB) is relatively small and expensive, RIMAC relieves RAID controllers from caching data that is inclusive to a larger storage cache. In SectionS 7.1 and 7.2, we studied the effects of storage cache size on performance and energy saving with a fixed 8 MB parity cache. In this section, we investigate the impact of parity cache size and the replacement algorithm on performance and energy consumption.

7.4.1 Effects of Parity Cache Size

We proportionally scale down the latest storage and RAID controller cache configuration pair (16 GB [2], 256 MB [3]) to the basic setting (64 MB, 1 MB) used in this study. Then, we vary the the size of parity cache from 1 MB to 32 MB and compare the performance improvement and energy savings to a baseline system with infinite storage cache. The results of RIMAC are normalized to those of the baseline system with infinite storage cache and commensurate parity cache.

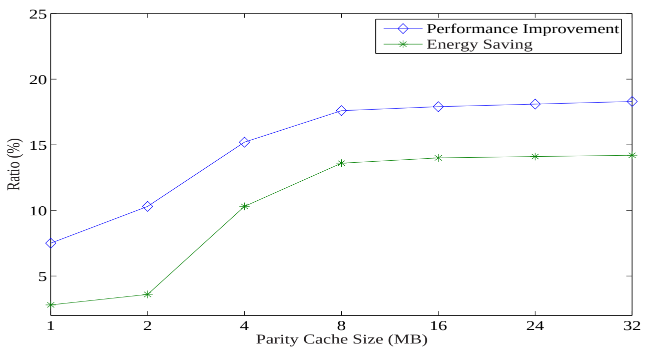

Figure 11: Normalized energy saving and performance improvement of RIMAC with different parity cache size.

Figure 11 shows the energy consumption and performance result for cello99. In one extreme, when the parity cache is small (e.g. 1 MB), both the energy savings and performance improvement of RIMAC are limited. With an appropriate parity cache size (e.g. $\geq$ 8 MB in our experiments), the chance to perform cache-level request transformations and power aware parity updates is greatly increased. As a result, both the energy savings and performance improvement approach the optimal results. In this case, further increase in parity cache size will not be a cost-effective solution. The effects of increasing parity cache size under other workloads are similar.

7.4.2 Effects Parity Cache Replacement Algorithm

Table 5 compares the performance of the second-chance cache replacement algorithm used in RIMAC with the basic LRU based replacement algorithm. For both schemes, the parity cache size is fixed at 8 MB for fair comparison. The size of the storage caches for cello99, TPC-D and SPC-SE are 64 MB, 128 MB and 256 MB, respectively. The parity cache hit ratio has been improved by 141.3% (from 9.2% to 22.2%) for cello99, 42.5% (from 10.6% to 15.1%) for TPC-D, and 75.7% from (3.3% to 5.8%) for SPC-SE. These results indicate that our second-chance parity cache replacement algorithm is good enough to provide parity stripes required by request transformation policies of RIMAC. It is also noted that the hit ratio of the read-exclusive SPC-SE workload is relatively lower than cello99 and TPC-D. This is because both cello99 and TPC-D are read and write mixed workloads. The parity update for write request also helps to increase the ratio of parity hit.

7.5 Energy Analysis

To show how RIMAC can both save energy and improve performance, we break down the energy consumed in each power state for two representative disks.

Algorithm	Cello99	TPC-D	SPC-SE
LRU	9.2%	10.6%	3.3%
2-Chance	22.2%	15.1%	5.8%

Table 5: Effect of Second-chance Parity Cache Replacement Algorithm vs. LRU in RIMAC

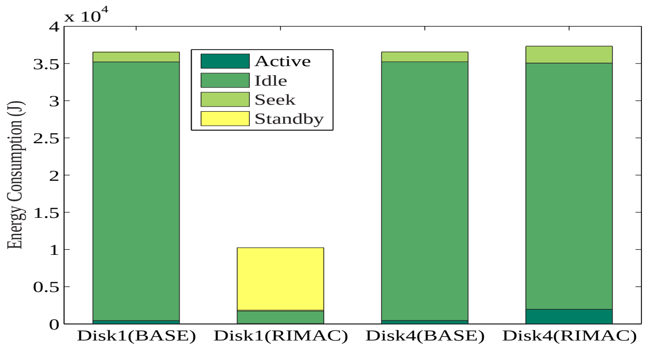

Figure 12: Energy consumption breakdown of cello99

Figure 12 shows the results of two representative disks — disk 1 and disk 4 for the cello99 trace with 64 MB total cache size for LRU and RIMAC respectively. Disk 1 consumes 1,671.5 J in the idle mode for RIMAC and 34,767.2 J for LRU. This is because most requests of disk 1 are transformed to other active disks in RIMAC. The fact that 8,386.4 J energy is consumed in the standby state on disk 1 indicates disk 1 is in the standby state for a much longer duration than disk 4.

Although most energy is consumed in the idle state in LRU baseline system, all disks can rarely find isolated idle periods longer than the break-even threshold. That is why we observe negligible energy consumption derived from standby for all the disks in baseline systems. Researchers from Pennsylvania State University [15, 23] made similar observations. They found that for an OLTP workload, TPC-H does not show any visible idle periods longer than 1 second and TPC-C only shows less than 1% of idle time longer than 2 seconds.

Given our request transformation policies, short idle periods can be accumulated into longer idle periods without incurring much overhead. The effects of request transformation policies on energy can be observed from the small increase in seek and active portions, amounting to 1,510.5 J and 918.7 J on disk 4 respectively. However, compared with 24,709.3 J energy saved on disk 1, overall energy consumption is dramatically reduced for the whole RAID-based storage system.

7.6 Scalability

Although RAID-5 can be as large as hundreds of disks in theory, the reliability issue prohibits us from building a large scale RAID-5. The size of commonly used RAID-5 as building blocks in today's data centers are 4, 5, 6, and 8 disks. In this section, we quantify the energy savings of our solutions on 4, 5, 6 and 8 disk RAID-5s. Due to space limitations, we only present the results of cello99 with a 64 MB total cache size.

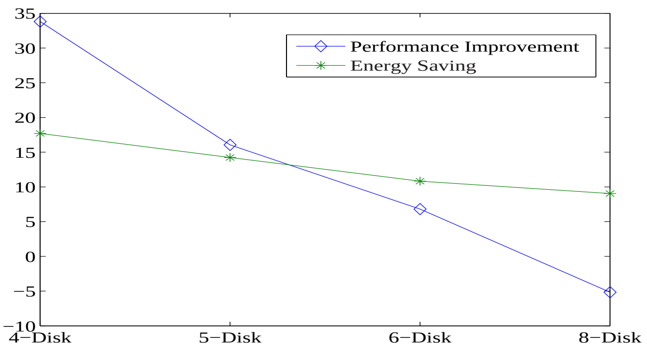

Figure 13: Energy saving and performance improvement of RIMAC when scaling the number of disks from 4 to 8.

Figure 13 shows the results of energy savings and performance improvement on average response times for 4, 5, 6 and 8 disk RAID-5 configurations. The results shown are normalized to a baseline system with infinite storage cache and commensurate parity cache. RIMAC with a four-disk RAID-5 can save 17.7% in energy consumption and 33.8% in average response time compared with the baseline system using LRU. Even though the RAID size increases to 6, RIMAC still achieves 10.8% energy savings and 6.8% performance improvement. Because RIMAC needs more accesses to do a request transformation when the size of RAID-5 further increases, we observe 5.2% performance degradation in an 8-disk RIMAC configuration, but we still realize 9.1% energy saving. Based on these results, we can conclude that RIMAC strikes a good tradeoff between energy-efficiency and high-performance by applying the power-aware request transformation and parity update optimization techniques.

8. CONCLUSION

In this article, we develop a redundancy-based, two-level I/O cache architecture called RIMAC to significantly reduce energy costs and simultaneously improve performance for RAID-based storage systems. RIMAC exploits inherent redundancy to perform power-aware request transformations for reads and writes *both* in two-level I/O cache and on physical disks. Comprehensive simulation results with several real-life server traces show that RIMAC outperforms existing solutions by up to 33% in terms of energy savings and up to 30% in terms of average response time.

9. ACKNOWLEDGEMENTS

We thank Anastassia Ailamaki for her excellent shepherding that improved the quality of this paper, and the anonymous reviewers for their thoughtful suggestions. We also thank Hewlett-Packard Labs Storage Systems Department for providing storage traces used in this paper. This work is supported in part by the US National Science Foundation under grants CNS-0509480 and CCF-0429995, the US Department of Energy Early Career Principal Investigator Award DE-FG02-05ER25687.

10. REFERENCES

[1] Dell/EMC SAN Configurations. http://www.dell.com/downloads/global/power/ps2q4-022.pdf/.

[2] IBM TotalStorage DS8000 Overview. http://www-1.ibm.com/servers/storage/disk/ds8000/.

[3] Intel RAID Controller. http://www.intel.com/design/servers/RAID/srcu42e/index.htm/.

[4] Storage Performance Council-Search Engine Traces. http://prisms.cs.umass.edu/repository/walk.php?cat=Storage/.

[5] More Power Needed. Energy User News, November 2002.

[6] Powerpc 750 microprocessors data sheet. http://www-306.ibm.com/chips/techlib/techlib.nsf/products/PowerPC_750_Microprocessor, Sep 2002.

[7] Service level agreement in the data center. http://www.sun.com/blueprints/0402/sla.pdf, April 2002.

[8] J. S. Bucy and G. R. Ganger. The disksim simulation environment version 3.0 reference manual. Technical Report CMU-CS-03-102, Carnegie Mellon University, School of Computer Science, Jan. 2003.

[9] E. V. Carrera, E. Pinheiro, and R. Bianchini. Conserving disk energy in network servers. In *Proceedings of the 2003 International Conference on Supercomputing (ICS-03)*, pages 86–97, New York, June 23–26 2003. ACM Press.

[10] P. M. Chen and E. K. Lee. Striping in a RAID level 5 disk array. In *Proceedings of ACM International Conference on Measurement and Modeling of Computing Systems (SIGMETRICS'95)*, 1995.

[11] Z. Chen, Y. Zhang, Y. Zhou, H. Scott, and B. Schiefer. Empirical evaluation of multi-level buffer cache collaboration for storage system. In *Proceedings of ACM International Conference on Measurement and Modeling of Computing Systems (SIGMETRICS'05)*, June 2005.

[12] Z. Chen, Y. Zhou, and K. Li. Eviction-based cache placement for storage caches. In *Proceedings of the USENIX Annual Technical Conference, General Track 2003*, pages 269–281, 2003.

[13] D. Colarelli and D. Grunwald. Massive arrays of idle disks for storage archives. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–11, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[14] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. DRPM: dynamic speed control for power management in server class disks. In D. DeGroot, editor, *Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA-03)*, volume 31, 2 of *Computer Architecture News*, pages 169–181, New York, June 9–11 2003. ACM Press.

[15] S. Gurumurthi, J. Zhang, A. Sivasubramaniam, M. Kandemir, H. Franke, N. Vijaykrishnan, and M. J. Irwin. Interplay of energy and performance for disk arrays running transaction processing workloads. In *Performance Analysis of Systems and Software (ISPASS)*, pages 123–132, Mar. 2003.

[16] J. L. Hennessy and D. A. Patterson. *Computer Architecture — A Quantitative Approach.* Morgan Kaufmann Publishing Company, 2003.

[17] H. Huang, W. Hung, and K. G. Shin. Fs2: Dynamic data replication in free disk space for improving disk performance and energy consumption. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP'05)*, October 2005.

[18] S. Irani, S. Shukla, and R. Gupta. Competitive analysis of dynamic power management strategies for systems with multiple power saving states. Technical report, UCI-ICS, 2001.

[19] D. Li and J. Wang. EERAID: Energy-efficient Redundant and Inexpensive Disk Array. In *Proceedings of the 11th ACM SIGOPS European Workshop*, September 20-22 2004.

[20] N. Megiddo and D. S. Modha. Arc: A self-tuning, low overhead replacement cache. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST 03)*, San Franciso, CA, 2003.

[21] D. Patterson et al. RAID: Redundant arrays of inexpensive disks. In *19 ACM SIGMOD Conf. on the Management of Data, Chicago*, June 1988.

[22] E. Pinheiro and R. Bianchini. Energy conservation techniques for disk array-based servers. In *Proceedings of the 18th International Conference on Supercomputing*, pages 68–78, June 26 - July 01 2004.

[23] A. S. V. Natarajan, S. Gurumurthi. Is traditional power management + prefetching == drpm for server disks? In *Eighth Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW-8)*, San Francisco, CA, February 2005.

[24] T. M. Wong and J. Wilkes. My cache or yours? making storage more exclusive. In *Proceedings of the USENIX Annual Technical Conference, General Track 2002*, pages 161–175, 2002.

[25] R. Youssef. RAID for mobile computers. Master's thesis, Carnegie Mellon University Information Networking Institute, Aug. 1995. Available as INI-TR 1995-3.

[26] Y. Zhou, J. Philbin, and K. Li. The multiqueue replacement algorithm for second level buffer caches. In *Proceedings of the USENIX Annual Technical Conference, General Track 2001*, Boston, Massachusetts, June 2001.

[27] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes. Hibernator: Helping disk arrays sleep through the winter. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP'05)*, October 2005.

[28] Q. Zhu, F. M. David, C. F. Devaraj, Z. Li, Y. Zhou, and P. Cao. Reducing energy consumption of disk storage using power-aware cache management. In *Tenth International Symposium on High Performance Computer Architecture (HPCA-10)*, Madrid, Spain, Feb. 14–18, 2004.

[29] Q. Zhu, A. Shankar, and Y. Zhou. Power-aware storage cache replacement algorithms. In *Proceedings of the International Conference on Supercomputing (ICS'04)*, pages 79–88, June 26 - July 01 2004.